Chapter 2
Embedded Memory Architecture for Low-Power Application Processor

Hoi Jun Yoo and Donghyun Kim

2.1 Memory Hierarchy

2.1.1 Introduction

Currently, the state-of-the-art high-end processors operate at 3–4 GHz frequency whereas even the fastest off-chip memory operates at just around 600 MHz [1–6]. In decades, along with advances in processor technology, the speed gap between processors and memories has become intolerably large [7], and this speed gap has driven the processor designers to introduce a memory hierarchy into the processor architecture. For processors, it is ideal to have indefinitely large memory with no access latencies [8]. However, implementing large-capacity memory with fast operation speed is infeasible due to the physical limitations of the electrical circuits. Thus, the capacity is usually traded off with the operation speed in memory designs. For example, on-chip L1 caches are able to operate as fast as the state-of-the-art processor cores but have at most few kilobytes capacity. On the other hand, off-chip DRAMs are capable of storing few gigabytes though their operation frequencies are just around hundreds of megahertz.

The memory hierarchy is an arrangement of different types of memories with different capacities and operation speeds to approximate the ideal memory behavior in a cost-efficient way. The idea of memory hierarchy comes from observing two common characteristics of the memory accesses in the wide range of programs, namely temporal locality and spatial locality. When a program accesses a certain data address repeatedly for a while, it is temporal locality. Spatial locality means that the memory accesses occur within a small region of memory for a short duration. Due to these localities, embedding a small but fast memory is sufficient to provide a processor with frequently required data for a short period of time. However,

H.J. Yoo (✉)
KAIST

K. Zhang (ed.), *Embedded Memories for Nano-Scale VLSIs*, Series on Integrated Circuits and Systems, DOI 10.1007/978-0-387-88497-4_2,

large-capacity memory to store the entire working set of a program and other necessary data such as the operating system is also necessary. In this case, the former is usually an L1 cache and the latter is generally realized by external DRAMs or hard disk drives in conventional computer systems. Since the speed difference between these two memories is at least more than four orders of magnitude, more levels of the memory hierarchy are required to hide and reduce long access latencies resulting from the small number of levels in the memory hierarchy. In typical computer systems, more than four levels of the memory hierarchy are widely adopted, and a memory at the higher level is realized as a smaller and faster memory than those of the lower levels. Figure 2.1 describes typical arrangement of the memory hierarchy.

2.1.2 Advantages of the Memory Hierarchy

The advantage of adopting the memory hierarchy is threefold. The first advantage is to reduce cost of implementing a memory system. In many cases, faster memories are more expensive than slower memories. For example, SRAMs require higher cost per unit storage capacity than DRAMs because a 6-transistor cell in the SRAMs consumes more silicon area than a single transistor cell of the DRAMs. Similarly, DRAMs are more costly than hard disk drives or flash memories for the same capacity. Flash memory cells consume less silicon area and platters of the hard disk drives are much cheaper than silicon die in a mass production. A combination of different types of memories in the memory system enables a trade-off between performance and cost. By storing infrequently accessed data in the slow but low-cost memories, the overall system cost can be reduced.

The second advantage is an improved performance. Without the memory hierarchy, a processor should directly access the lowest level memory that operates very slowly and contains all required data. In this case, every memory access results in

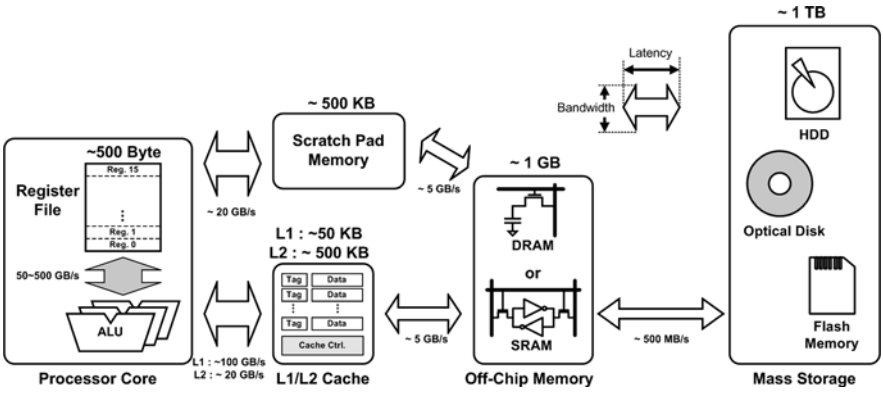


Fig. 2.1 Memory hierarchy

processor stalls to wait for the required data to be available from the memory. Such drawback is resolved by embedding a small memory that runs as fast as a processor core inside the chip. By maintaining an active working set inside the embedded memory, no processor stalls due to memory accesses occur as long as a program is executed within the working set. However, the processor could stall when a working set replacement is performed. This overhead can be reduced by pre-fetching the next working set in the additional in-between level of memory which is easier to access than the lowest level memory. In this way, the memory hierarchy builds up so that the number of levels and types of memories in the memory hierarchy are properly adjusted to minimize average wait cycles for memory accesses. However, finding the optimum configuration of the memory hierarchy requires sophisticated investigation of target application, careful consideration of processor core features, and exhaustive design space exploration. Therefore, design of the memory hierarchy has been one of the most active research fields from the emergence of the computer architecture.

The third advantage of the memory hierarchy is reducing power consumption of a memory system. Accessing an external memory consumes more power than accessing an on-chip memory because off-chip wires have larger parasitic capacitance due to their bigger dimensions. Charging and discharging such large parasitic capacitors result in significant power overhead of off-chip memory accesses. Adopting the memory hierarchy is advantageous to reduce the number of external memory transactions, thus also reducing the power overhead. In a program execution, dynamic data are divided into two categories. One of them is temporary data used to calculate and produce output data of a program execution, and the other is result data that are used by other programs or I/O devices. The result data need to be stored in an off-chip memory such as a main memory or hard disk drive for later reuse of the data. However, temporary data do not need to be stored outside of the chip. Embedding on-chip memories inside the processor enables keeping the temporary data inside the chip during program execution. This reduces the chance of reading or writing of the temporary data in the external memory, which is very costly in power consumption.

2.1.3 Components of the Memory Hierarchy

This section briefly describes different types of memories that construct typical memory hierarchy in conventional computer architectures.

2.1.3.1 Register File

A register file constructs the highest level of the memory hierarchy. A register file is an array of registers embedded in the processor core and is tightly coupled to datapath units to provide an immediate storage for the operands to be calculated. Each entry of the register file is directly accessible without address calculation in the arithmetic and logic unit (ALU) and is defined in an instruction set architecture

(ISA) of the processor. The register file is usually implemented using SRAM cells, and the I/O width is determined to match the datapath width of the processor core. The register file usually has larger number of read ports than conventional SRAMs to provide an ALU with required number of operands in a single cycle. In the case of superscalar processors or very long instruction word (VLIW) processors, the register file is equipped with more than two write ports to support multiple register writes resulting from parallel execution of multiple instructions. Typical number of entries in a register file is around a few tens, and the operation speed is the same as the processor core in most cases.

2.1.3.2 Cache

A cache is a special type of memory that autonomously pre-fetches a subset of temporary duplicated data from lower levels of the memory hierarchy. The caches are the principal part of the memory hierarchy in most computer architectures, and there is a hierarchy among the caches as well. Level 1 (L1) and level 2 (L2) caches are widely adopted and level 3 (L3) cache is usually optional. The L1 cache has the smallest capacity and the lowest access latency. On the other hand, the L3 cache has the largest capacity and the longest access latency. Because the caches maintain duplicated copy of data, cache controllers to manage consistency and coherency schemes are also required to prevent the processing core fetching outdated copies of the data. In addition, the cache includes a tag memory to look up which address regions are stored in the cache.

2.1.3.3 Scratch Pad Memory

A scratch pad memory is an on-chip memory under the management of a user program. The scratch pad memory is usually adopted as a storage of frequently and repeatedly accessed data to reduce the external memory transactions. The size of the scratch pad memory is in the range of tens or hundreds of kilobytes and its physical arrangements, such as number of ports, bank, and cell types, are application-specific. The scratch pad memory is generally adopted for real-time embedded systems to guarantee the predictability in program execution time. In cache-based systems, it is hard to guarantee worst execution time, because behaviors of caches are not under the control of a user program and vary dynamically depending on the dynamic status of the memory system.

2.1.3.4 Off-Chip RAMs

The random access memory (RAM) is a type of memory that allows a read/write access to any address in a constant time. The RAMs are mainly divided into dynamic RAM (DRAM) and static RAM (SRAM) according to their internal cell structures. The term RAM does not specify a certain level in the memory hierarchy, and most of the memories such as cache, scratch pad memory, and register files in the memory hierarchy are classified as RAMs. However, a RAM implemented in a separate

package usually specifies a certain level in the memory hierarchy, which is lower than the caches or scratch pad memories. In the perspective of a processor, such RAMs are referred to as off-chip RAMs. The process technologies used to implement off-chip RAMs are optimized to increase memory cell density rather than fast logic operation. The off-chip SRAMs are used as L3 caches or main memory of handheld systems due to their fast operation speed and low-power consumption compared to the off-chip DRAMs. The off-chip DRAMs are used as main memory of a computer system because of their large capacity. In the DRAMs, the whole working set of a program that does not fit into the on-chip cache or scratch pad memory is stored. The DRAMS are usually sold in a single or dual in-line memory module (SIMM or DIMM) that is assembled with a number of DRAM packages on a single printed circuit board (PCB) to achieve large capacity up to few gigabytes.

2.1.3.5 Mass Storages

The lowest level of the memory hierarchy consists of mass storage devices such as hard disk drives, optical disk, and back-up tapes. The mass storage devices have the longest access latencies in the memory hierarchy but provide the largest capacity sufficient to store entire working set as well as other peripheral data such as operating system, device drivers, and result data of program executions for future use. The mass storage devices are usually non-volatile memories able to retain internal data without power supply.

2.2 Memory Access Pattern Related Techniques

In this and following sections, low-power techniques applicable to embedded memory system are described based on the background knowledge of the previous section. First, this section covers memory architecture design issues regarding memory access patterns.

If the system designers understand the memory access pattern of the system operation and it is possible to modify the memory interface, the system performance as well as power consumption can be enhanced by removing or reducing unnecessary memory operations. For some applications having predictable memory access patterns, it is possible to improve effective memory bandwidth with no cost overhead by understanding the intrinsic characteristics of the memory device. And sometimes the system performance is increased by modifying the memory interface. The following case studies show how the system performance and power efficiency are enhanced by understanding the memory access pattern.

2.2.1 Bank Interleaving

When accessing a DRAM, a decoded row address activates a word line and corresponding bit-line sense amplifiers so that the cells connected to the activated word

line are ready to transfer or accept data. And then the column address decides which cell in the activated row is connected to the data-bit (DB) sense amplifier or write driver. After accessing data, data signals such as bit lines and DB lines are pre-charged for the next access. Thus, a DRAM basically needs "row activation," "read or write," and "pre-charge" operations to access data. The sum of their operation times decides the access time. The "row activation" and "pre-charge" operations occupy most of the access time and they are not linearly shrunk according to the process downscaling, whereas the operation time of "read or write" is sufficiently reduced to be completed in one clock cycle even with the faster clock frequency of smaller scale process technologies, as shown in Fig. 2.2. In the case of the cell array arranged in a single bank, these operations should be executed sequentially and cannot be overlapped. On the other hand, by dividing the cell array into two or more banks, it is possible to scatter sequential addresses into multiple banks by modulo *N* operations as shown in Fig. 2.3(b), where *N* is the number of memory

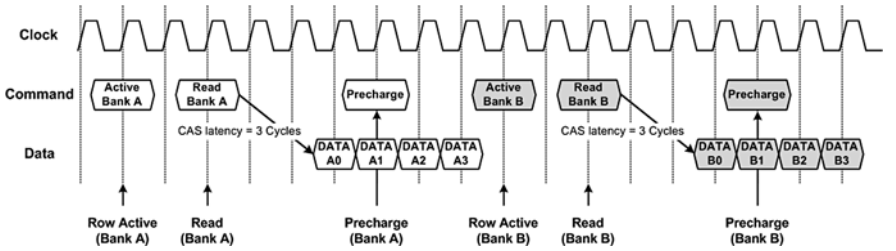


Fig. 2.2 Timing diagram of DRAM read operations without bank interleaving

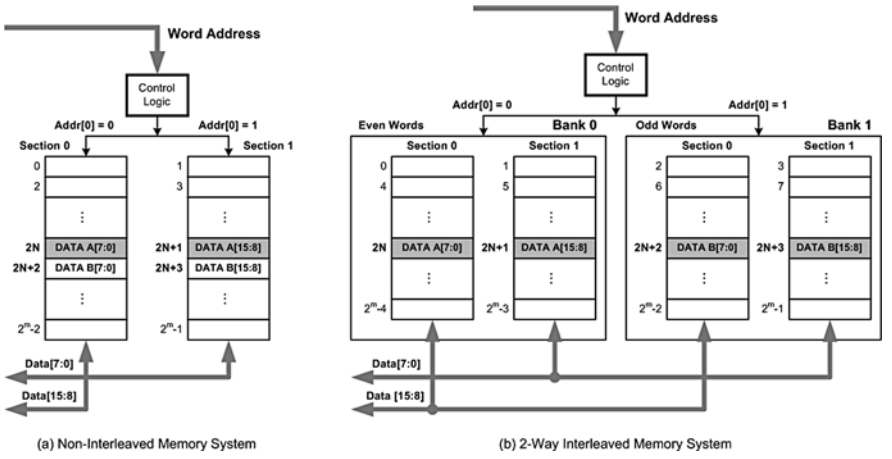


Fig. 2.3 Structures of non-interleaved and interleaved memory systems

banks. And this contributes to hiding the "row activation" or "pre-charge" time of the cell array.

Bank interleaving exploits the independency of the row activations in the different memory banks. Bank is the unit of the cell array which shares the same row and column addresses. By dividing the memory cells into multiple banks, we can obtain the following advantages [9, 10]:

- It hides the amount of time to pre-charge or activate the arrays by accessing one during pre-charging or activating the others, which means that high bandwidth is obtained with low-speed memory chip.
- It can save the power consumption by activating only a subset of cell array at a time.
- It keeps the size of each cell array smaller and limits the number of row and column address pins, and this results in cost reduction.

Figure 2.3 shows the memory configurations with and without bank interleaving. The configuration in Fig. 2.3(a) consists of two sections with each section covering 1-byte data. After it accesses one word, namely addresses 2*N* and 2*N*+1, it needs pre-charge time to access the next word, addresses 2*N*+2 and 2*N*+3. And the row activation for the next access cannot be overlapped. The configuration in Fig. 2.3(b), however, consists of two banks and it can activate the row for the addresses 2*N*+2 and 2*N*+3 while the row for the addresses 2*N* and 2*N*+1 is pre-charged.

Figure 2.4 shows the timing diagram of read operation with interleaved memory structure. Figures 2.2 and 2.4 both assume that column address strobe (CAS) latency is 3 and burst length is 4. On comparing with Fig. 2.2, interleaved memory structure generates 8 data in 12 clock cycles, while non-interleaved memory structure needs additional 5 clock cycles.

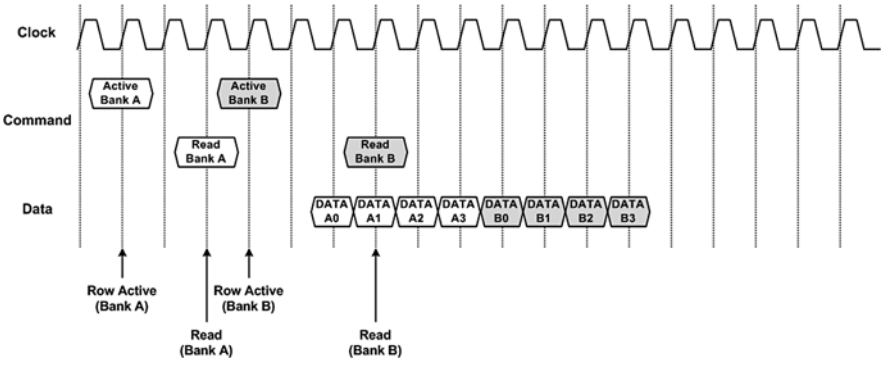


Fig. 2.4 Timing diagram of DRAM read operations with bank interleaving

2.2.2 Address Alignment Logic in KAIST RAMP-IV

In 3D graphics applications, rendering engine requires large memory bandwidth to render high-quality 3D images in real time. To obtain large memory bandwidth, the memory system needs wide data bus or fast clock frequency. Otherwise, we can virtually enlarge the bandwidth by reusing data which had been accessed.

The address alignment logic (AAL) [11] in the 3D rendering engine exploits the access pattern of the texture pixel (texel) from the texture memory. Generally a pixel is calculated using four texels as shown in Fig. 2.5. And corresponding four memory accesses are required. Observing the address of the four texels, they are normally neighbored to each other because of their spatial correlation. If the rendering engine is able to recognize which address it had accessed before, it does not need to access it again, because it has already fetched the data. Figure 2.6(a) shows the block diagram of the AAL. It checks the texel addresses spatially and temporally. If the address had been accessed before and is still available in the logic block, it does not generate the data request to texture memory. Although the additional check operation increases the cycle time, the average number of the texture memory accesses is reduced to less than 30% of the memory access count without the AAL. Figure 2.6(b) shows the energy reduction by the AAL; 68% of the total energy consumption is reduced by understanding and exploiting the memory access pattern.

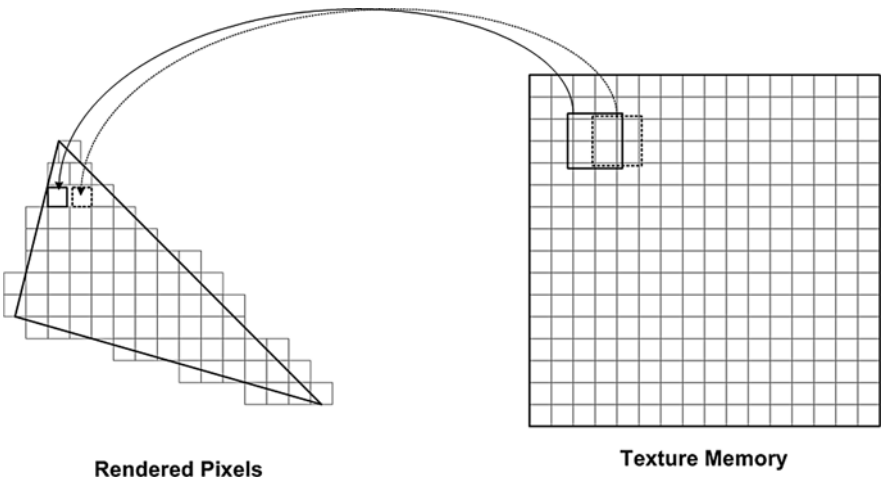


Fig. 2.5 Pixel rendering with texture mapping

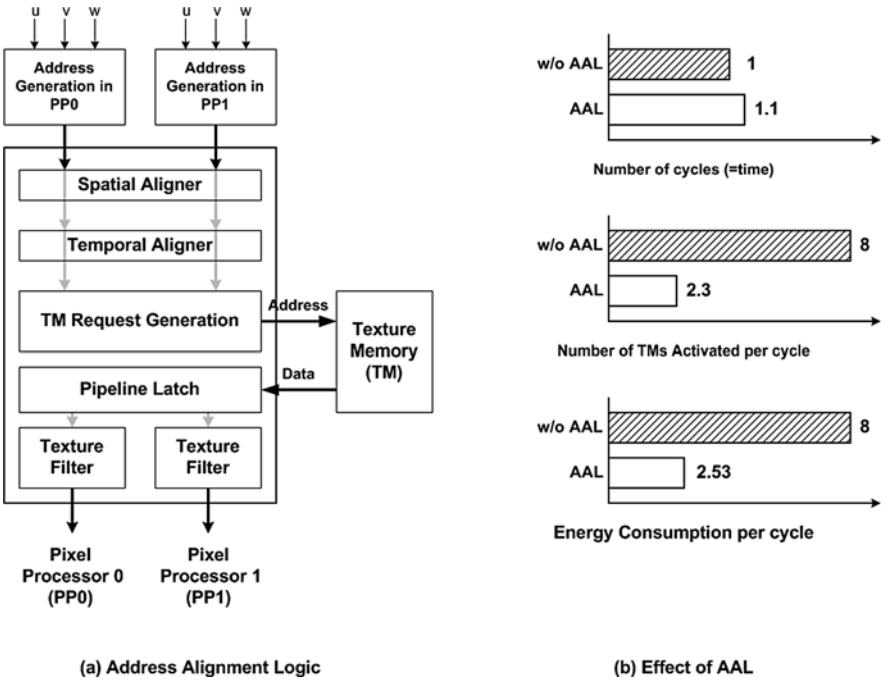


Fig. 2.6 Block diagram of address alignment logic (AAL) (**a**) and its effects (**b**)

2.2.3 Read–Modify–Write (RMW) DRAM

Read–modify–write (RMW) is a special case in which a memory location is first read and then re-written again. It is useful for 3D graphics rendering applications, especially for frame buffer and depth buffer. Frame buffer stores an image data to be displayed, and depth buffer stores the depth information of each pixel. Both of them are accessed by 3D graphics processor, and data are compared and modified. In the depth comparison operations, for example, depth buffer data are accessed and the depth information is modified. If the depth information of stored pixel is screened by newly generated pixel, it needs to be updated. And the frame buffer is refreshed every frame. Both memory devices require three commands: read, modify, and write. From the memory point of view, modify is just waiting. If the memory consists of DRAM cells, it needs to carry out "Row Activation–Read–Pre-charge–Nop (Wait)–Row Activation–Write–Pre-charge" sequences to the same address. If it supports RMW operations, the command sequence can be reduced to "Row Activation–Read–Wait–Write–Pre-charge" as shown in Fig. 2.7, which is compact with no redundant operations. The RMW operation shows that the data bandwidth and control complexity can be reduced by modifying the command sequences regarding the characteristics of memory accesses.

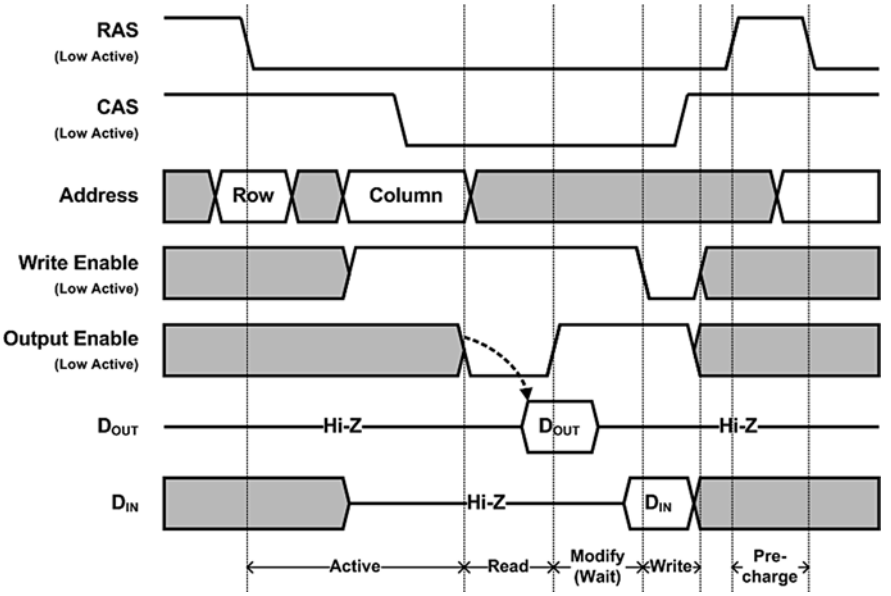


Fig. 2.7 Read–modify–write operation timing diagram

2.3 Embedded Memory Architecture Case Studies

In the design of low-power system-on-chip (SoC), architecture of the embedded memory system has significant impact on the power consumption and overall performance of the SoC. In this section, three embedded memory architectures are covered as case studies. The first example is a Marvell PXA 300 processor which represents a general-purpose application processor. The second example is an IMAGINE processor aimed at removing bandwidth bottleneck in stream processing applications. The last example is the memory-centric network-on-chip (NoC) which adopts co-design of memory architecture and NoC for efficient execution of pipelined tasks.

2.3.1 PXA300 Processor

The PXA series processors were first released by Intel in 2002. The PXA processor series were sold to Marvell technology group in 2006, and PXA3XX series processors are in mass production currently. The PXA300 processor is a general-purpose SoC which incorporates a processor core and other peripheral hardware blocks [12]. The processor core based on an ARM instruction set architecture (ISA) is integrated for general-purpose applications. The SoC is also featured with a 2D graphic processor, video/JPEG acceleration hardware, memory controllers, and an LCD controller. Figure 2.8 shows simplified block diagram of the PXA300 processor [13]. As

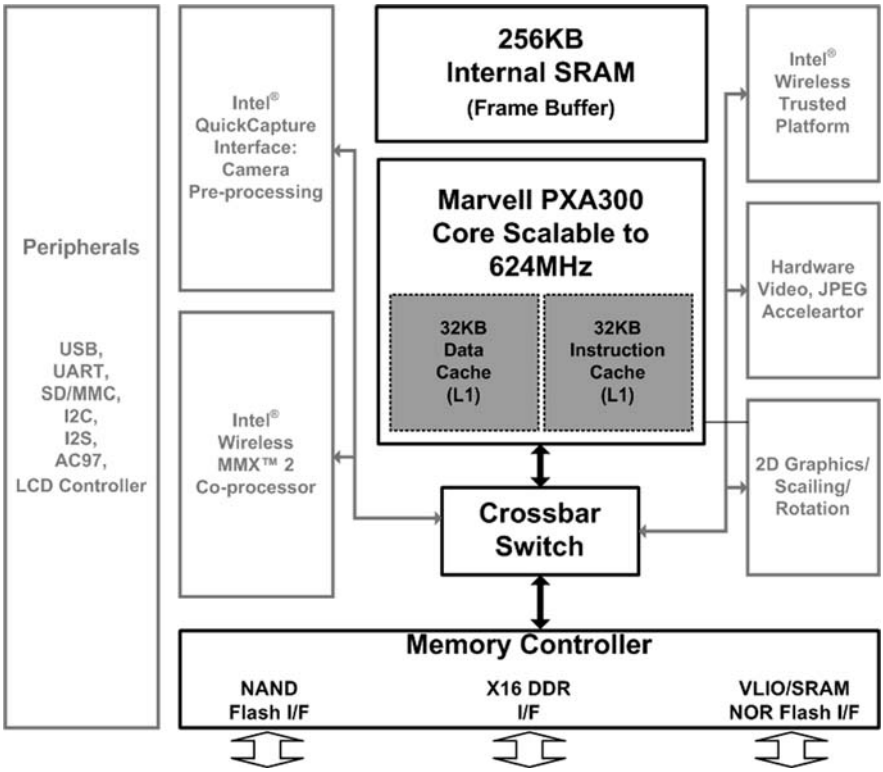


Fig. 2.8 Block diagram of PXA300 processor

shown in Fig. 2.8, the memory hierarchy of the PXA300 is rather simple. The processor core is equipped with L1 instruction/data caches, and both caches are sized to 32 KB. Considering relatively small difference in the operation speed of processor core and the main memory provided by an off-chip double data rate (DDR) SDRAM, absence of an L2 cache is a reasonable design choice. The operation speed of the DDR memory is in the range of 100–200 MHz, and the clock frequency of the processor core is designed to be just around 600 MHz for low-power consumption. Besides the L1 caches, a 256 KB on-chip SRAM is incorporated to provide frame buffer for video codec support. Because the frame buffer requires continuous update of its context and consumes large memory bandwidth, integrating the on-chip SRAM and LCD controller contributes to reducing the external memory transactions. The lowest level of the memory hierarchy consists of flash memories such as NAND/NOR flash memories and secure digital (SD) cards to adapt for handheld devices. Since the PXA300 processor is targeted for general-purpose applications, it is hard to tailor the memory system for low-power execution of a specific application. Therefore, the memory hierarchy of the PXA300 processor is designed similar to those of conventional computer systems with some modifications appropriate for

handheld devices. Instead, low-power technique is applied for the entire processor so that operation frequency of the chip is varied according to the workload.

2.3.2 Imagine

In contrast to the general-purpose PXA300 processor, the IMAGINE is more focused on applications having streamed data flow [14, 15]. The IMAGINE processor has customized memory architecture to maximize the available bandwidth among on-chip processing units that consist of 48 ALUs. The memory architecture of the IMAGINE is tiered into three levels so that the memory hierarchy leverages the available bandwidth from the outside of the chip to the internal register files. In this section, the architecture of the IMAGINE processor is briefly described, and then the architectural benefits for efficient stream processing are discussed.

Figure 2.9 shows the overall architecture of the IMAGINE processor. The processor consists of a streaming memory system, a 128 KB streaming register file (SRF), and 48 ALUs divided into 8 ALU clusters. In each ALU cluster, 17 local register files (LRFs) are fully connected to each other through a crossbar switch and the LRFs provide operands for the 6 ALUs, a scratch pad memory, and a communication unit as shown in Fig. 2.10. The lowest level of the memory hierarchy in the IMAGINE is the streaming memory system, which manages four independent 32-bit wide SDRAMs operating at 167 MHz to achieve 2.67 GB/s bandwidth between external memories and the IMAGINE processor. The second level of the memory hierarchy consists of the SRF including a 128 KB SRAM divided into 1024 blocks. All accesses to the SRF are performed through 22 stream buffers and they

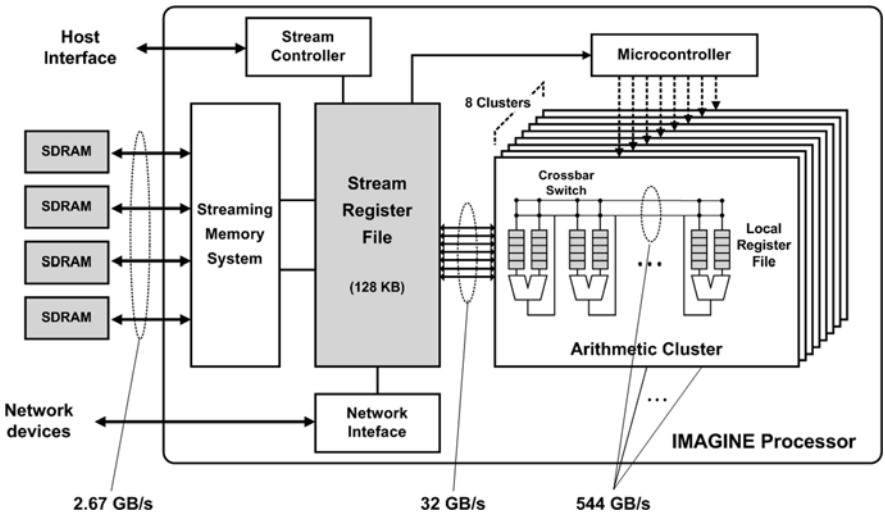


Fig. 2.9 Block diagram of the IMAGINE processor

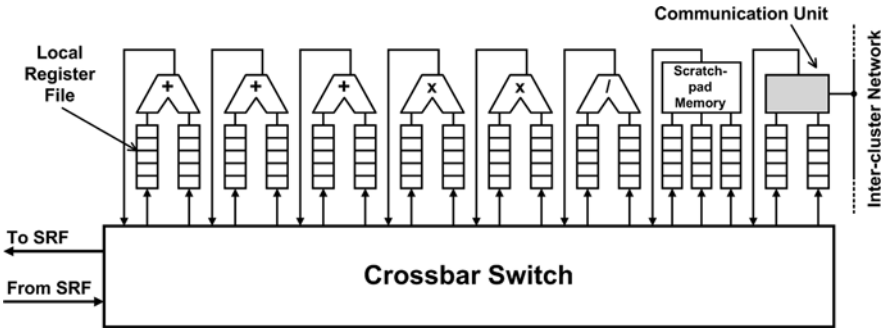


Fig. 2.10 Block diagram of an ALU cluster in the IMAGINE processor

are partitioned into 5 groups to interact with different modules of the processor. By pre-fetching the SRAM data into the stream buffers or utilizing the stream buffers as write buffers, the single-ported SRAM is virtualized as a 22-ported memory, and the peak bandwidth between the SRF and the LRF is 32 GB/s when the IMAGINE operates at 500 MHz. In this case, the 32 GB/s bandwidth is not a sustained bandwidth but a peak bandwidth because the stream buffers for the LRF accesses are managed in time-multiplexed fashion. Finally, the first level of the memory hierarchy is realized by the number of LRFs and crossbar switches. As shown in Fig. 2.10, the fully connected 17 LRFs in each ALU cluster provide a vast amount of bandwidth among the ALUs in each cluster. In addition, eight ALU clusters are able to communicate with each other throughout the SRF or inter-cluster network. The aggregated inter-ALU bandwidth among the 48 ALUs of the 8 ALU clusters reaches up to 544 GB/s.

The architectural benefits of the IMAGINE are found by observing characteristics of the stream processing applications. Stream processing refers to performing series of computation kernels repeatedly on a streamed data flow. In practical designs, the kernel has a set of instructions to be executed for a certain type of function. In stream processing applications such as video encoding/decoding, image processing, and object recognition, major portion of the input data is in the form of video streams. To process vast amount of pixels in a video stream with sufficiently high frame rate, stream processing usually requires intensive computation. Fortunately, in many applications, it is possible to process separate regions of the input data stream independently, and this allows exploiting data parallelism for stream processing. In addition, little reuse of input data and producer consumer locality are the other characteristics of stream processing.

The architecture of the IMAGINE is designed to take advantage of knowledge about the memory access patterns and to exploit intrinsic parallelism of stream processing. Since fixed set of kernels are repeatedly performed on an input data stream, memory access patterns of stream processing are predictable and scheduling of the memory accesses from the multiple ALU is also possible. Therefore, pre-fetching data from the lower level of memory hierarchy, i.e., the streaming memory system

or SRF, is effective for hiding latencies of accessing the off-chip SDRAMS from the ALU clusters. In the IMAGINE, all data transfers are explicitly managed by the stream controller shown in Fig. 2.9. Once pre-fetched data are prepared in the SRF, the large 32 GB/s bandwidth between the SRF and the LRFs is efficiently utilized to provide the 48 ALUs with multiple data simultaneously. After that, background pre-fetch operation of the next data is scheduled while the ALU clusters are computing fetched data. However, in the case of general-purpose applications, large peak bandwidth of the IMAGINE is not always available because scheduling of data pre-fetching is impossible for some applications with non-predictable data access patterns.

Another aspect of the stream processing, data parallelism, is also considered in the architecture of the IMAGINE, hence the eight ALU clusters are integrated to exploit data parallelism. The eight clusters perform computations on a divided part of the working set in parallel, and the six ALUs in each cluster compute kernels in a VLIW fashion. Large bandwidth among the ALUs and LRFs is provided for efficient forwarding of the operands and reuse of partial data calculated in the process of computing the kernels. Finally, producer–consumer locality is the key characteristic of stream processing, which is practical for reducing external memory transactions. In stream processing, a series of computation kernels are executed on an input data stream and large amounts of intermediate data are transacted between the adjacent kernels. If these intermediate data are only produced by a specific kernel and only consumed by a consecutive kernel, there is a producer–consumer locality between the kernels. In this case, it is not necessary to share these intermediate data globally and to maintain them in the off-chip memory for later reuse. In the IMAGINE, the SRF provides temporary storage for such intermediate data, thus reducing external memory transactions. In addition the stream buffers facilitate parallel data transactions between the producer and the consumer kernels computed in parallel.

In summary, the IMAGINE is an implementation of the customized memory hierarchy based on the common characteristics of memory transactions in the stream applications. Regarding the predictability in the memory access patterns, the memory hierarchy is designed so that peak bandwidth is gradually increased from outside of the chip to the ALU clusters. The increased peak bandwidth is fully utilizable by explicit management of the data transactions and also practical for facilitating parallel executions of the eight ALU clusters. The SRF of the IMAGINE is designed to store intermediate data having producer–consumer locality, and this is useful for reducing power consumption because unnecessary off-chip data transactions can be reduced. The other feature helpful for low-power consumption is the LRFs in the ALU clusters which maintain frequently reused intermediate data close to the processing units.

2.3.3 Memory-Centric NoC

In this section, the memory-centric Network-on-Chip (NoC) [16, 17] is introduced as a more application-specific implementation of the memory hierarchy. A target

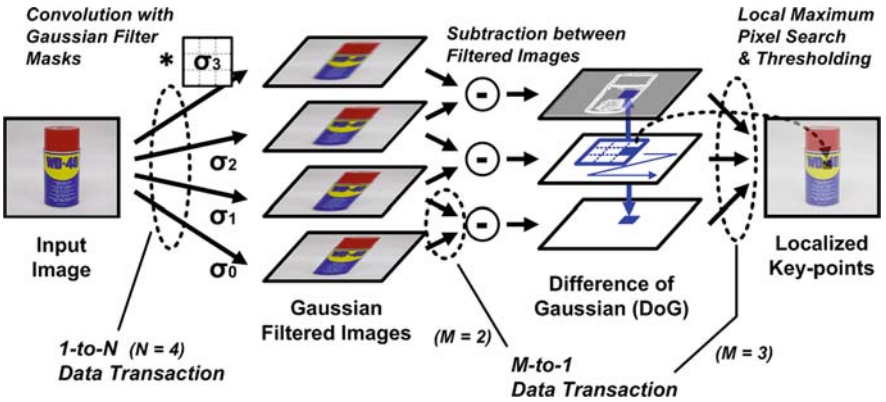
application of the memory-centric NoC is the scale-invariant feature transform (SIFT)-based object recognition. The SIFT algorithm [18] is widely adopted for autonomous navigation of mobile intelligent robots [19–22]. Due to vast amount of computation and limited power supply of the mobile robots, power-efficient computing of object recognition is demanded. The memory-centric NoC was proposed to achieve power-efficient object recognition by reducing external memory transactions of temporary data and overhead of data sharing in the multi-processor architecture. In addition, special-purpose memory is also integrated into the memory-centric NoC to further reduce power consumption by replacing complex operation with simple memory read operation. In this section, target application of the memory-centric NoC is described first to discover characteristics of the memory transactions. After that, architecture, operation, and benefits of the memory-centric NoC to implement power-efficient object recognition processor are explained.

2.3.3.1 SIFT Algorithm

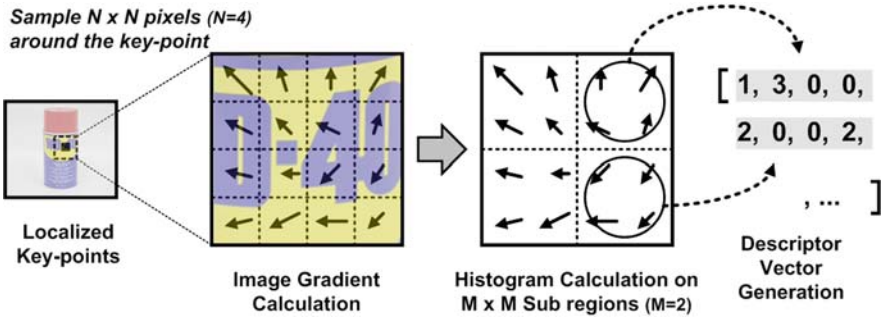
The scale-invariant feature transform (SIFT) object recognition [18] involves a number of image processing stages which repeatedly perform complex computations on the entire pixels of the input image. Based on the SIFT, points of human interest are extracted from the input image and converted into vectors that describe the distinctive features of the object. The vectors are then compared with the other vectors in the object database to find the matched object. The overall flow of the SIFT computation is divided into key-point localization and descriptor vector generation stages as shown in Fig. 2.11. For the key-point localization, Gaussian filtering with varying coefficients is performed repeatedly on the input image. Then, subtractions among the filtered images are executed to yield the difference of Gaussian (DoG) images. By performing the DoG operation, the edges of different scales are detected from the input image. After that, 3×3 search window is traversed over all DoG images to decide the locations of the key points by finding the local maximum pixels inside the window. The pixels having a local maximum value greater than a given threshold become the key points.

The next stage of the key-point localization is the descriptor vector generation. For each key-point location, $N \times N$ pixels of the input image are sampled first, and then the gradient of the sampled image is calculated. The sample size N is decided according to the DoG image where the key-point location is selected. Finally, a descriptor vector is generated by computing the orientation and magnitude histograms over $M \times M$ subregions of the sampled input image. The number of key points detected for each object is about a few hundreds.

As shown in Fig. 2.11, each task of the key-point localization consumes and produces a large amount of intermediate data, and the data should be transferred between the tasks. The data transaction between tasks has significant impact on the overall object recognition performance. Therefore, the memory hierarchy design should account for characteristics of the data transactions. Here, we note two important characteristics of the data transaction in the key-point localization stage of the SIFT calculation.



(a) Key-point localization



(b) Descriptor vector generation

Fig. 2.11 Overall flow of the SIFT computation

The first point is regarding the data dependency between the tasks. As illustrated in Fig. 2.11(a), the processing flow is completely pipelined; thus data transactions only occur between two adjacent tasks. This implies that the data transaction of the SIFT object recognition has producer–consumer locality as well, and the memory hierarchy should be adjusted for tasks, organizing a task-level pipeline. The second point is concerning the number of initiators and targets in the data transaction. In the multi-processor architecture, each task such as Gaussian filtering or DoG will be mapped to a group of processors, and the number of processors involved in each task can be adjusted to balance the execution time. For example, the Gaussian filtering in Fig. 2.11(a), having the highest computational complexity due to the 2D convolution, could use four processors to filter operations with different filter coefficients, whereas all of the DoG calculation is executed on a single processor. Due to the flexibility in task mapping, the resulting data of one processor is transferred to multiple processors of subsequent task or the results from multiple processors are transferred to one processor. This implies that the data transaction will occur in the forms of not only 1-to-1 but also 1-to-*N* and *M*-to-1, as shown in Fig. 2.11(a).

By regarding the characteristics of the data transactions discussed, the memory-centric NoC realizes the memory hierarchy that supports efficient 1-to-*N* and *M*-to-1 data transactions between the pipelined tasks. Therefore the memory-centric NoC facilitates configuring of variable types of pipelines in the multi-processor architectures.

2.3.3.2 Architecture of the Memory-Centric NoC

The overall architecture of the object recognition processor incorporating the memory-centric NoC is shown in Fig. 2.12. The main components of the proposed processor are the processing elements (PEs), eight visual image processing (VIP) memories, and an ARM-based RISC processor. The RISC processor controls the overall operation of the processor by initiating the task execution of each PE. After initialization, each PE fetches and executes an independent program for parallel execution of multiple tasks. The eight VIP memories provide communication buffers between the PEs and accelerate the local maximum pixel search operation. The memory-centric NoC is integrated to facilitate inter-PE communications by dynamically managing the eight VIP memories. The memory-centric NoC is composed of five crossbar switches, four channel controllers, and a number of network interface modules (NIMs).

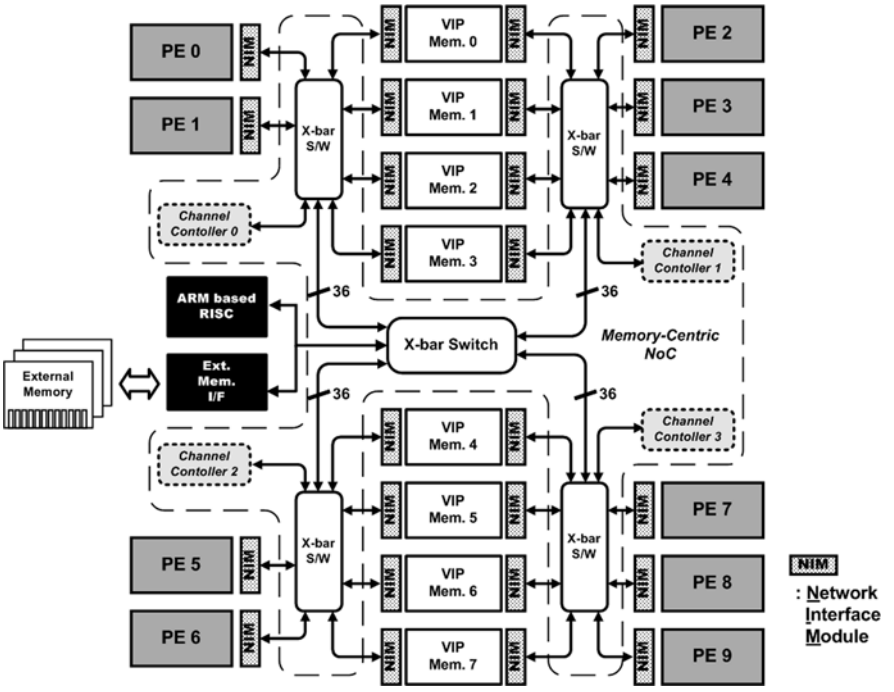


Fig. 2.12 Architecture of the memory-centric NoC

The topology of the memory-centric NoC is decided by considering the characteristics of the on-chip data transactions. For efficient support of the 1-to-*N* and *M*-to-1 data transactions shown in Fig. 2.11(a), using the VIP memory as a shared communication buffer is practical for removing the redundant data transfer when multiple PEs require the same data. Because the data flow through the pipelined tasks, each PE accesses only a subset of the VIP memories to receive the source data from its former PEs and send the resulting data to its following PEs. This results in localized data traffic, which allows tailoring of the NoC topology for low power and area reduction. There has been a research concerning power consumption and silicon area of the NoC in relation to NoC topologies [23], which concluded that a hierarchical star topology is the most efficient in case of interconnecting a few tens of on-chip modules with localized traffics. Therefore, the memory-centric NoC is configured in a hierarchical star topology instead of a regular mesh topology. By adopting a hierarchical star topology for the memory-centric NoC, the architecture of the proposed processor is able to be determined so that average hop counts between each PE and the VIP memories are reduced at the expense of a large direct PE-to-PE hop count, which is fixed to 3. This is also advantageous because most data transactions are performed between the PEs and the VIP memories, and direct PE-to-PE data transactions rarely occur. In addition, the VIP memory adopts dual read/write ports to facilitate short-distance interconnections between the ten PEs and the eight VIP memories. The NIMs are placed at each component of the processor to perform packet generation and parsing.

2.3.3.3 Memory-Centric NoC Operation

The operation of the memory-centric NoC is divided into two parts. The first part is to manage the utilization of the communication buffers, i.e., the VIP memories, between the producer and the consumer PEs. The other part is to support the memory transaction control after the VIP memory is assigned for the shared data transactions. The former operation removes the overhead of polling-available buffer spaces and the latter one reduces the overhead of waiting for valid data from the producer PE.

The overall procedure of the communication buffer management in the memory-centric NoC is shown in Fig. 2.13. Throughout the procedure, we assume that PE 1 is the producer PE and PEs 3 and 4 are consumer PEs. This is an example case of representing the 1-to-*N* (*N*=2) data transaction. The transaction is initiated by PE 1 writing an *open channel* command to the channel controller connected to the same crossbar switch (Fig. 2.13(a)). The *open channel* command is a simple memory-mapped write and transfers using a normal packet. In response to the *open channel* command, the channel controller reads the global status register of the VIP memories to check the utilization status. After selecting an available VIP memory, the channel controller updates the routing look-up tables (LUTs) in the NIMs of PEs 1, 3, and 4, so that the involved PEs read the same VIP memory for data transactions (Fig. 2.13(b)). The routing LUT update operation is performed by the channel controller sending the configuration (CFG) packets. At each PE, read/write accesses

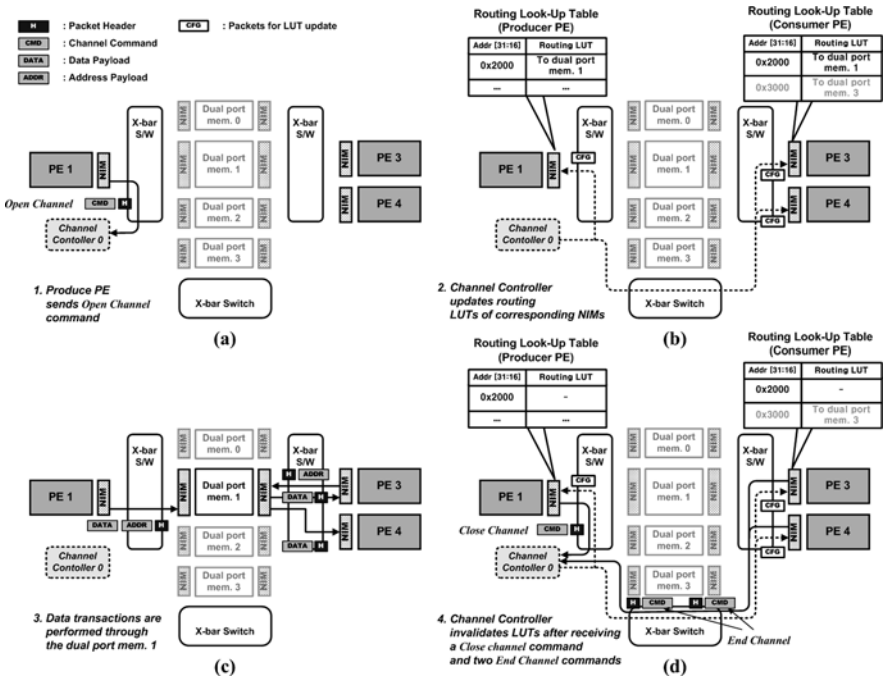


Fig. 2.13 Communication buffer management operation of the memory-centric NoC

for shared data transaction are blocked by the NIMs until the routing LUT update operation finishes. Once the VIP memory assignment is completed, a shared data transaction is executed using the VIP memory as a communication buffer. Read and write accesses to the VIP memory are performed using normal read/write packets that consist of an address and/or data fields (Fig. 2.13(c)). After the shared data transaction completes, PE 1 sends a *close channel* command, and PEs 2 and 3 send *end channel* commands to the channel controller. After that, the channel controller sends CFG packets to the NIMs of PEs 1, 3, and 4 to invalidate the corresponding routing LUT entries and to free up the used VIP memory (Fig. 2.13(d)).

From the operation of communication buffer management, efficient 1-to-*N* shared data transaction is clearly visible. Compared with the 1-to-1 shared data transaction, the required overhead is only sending additional (*N*–1) CFG packets at the start/end of the shared data transaction without making additional copy of shared data. In addition, an *M*-to-1 data transaction is also easily achieved by the consumer PE simply reading *M* VIP memories assigned to *M* producer PEs.

The previous paragraphs dealt with how the memory-centric NoC manages the utilization of VIP memories. In this paragraph, the memory transaction control scheme for efficient shared data transfer is explained. In the memory-centric NoC operation, no explicit loop is necessary to prevent consumer PEs reading the shared data too early before the producer PE writes valid data. To support the memory

transaction control, the memory-centric NoC tracks every write access to the VIP memory from the producer PE after the VIP memory is assigned to shared data transactions. This is realized by integrating a valid bit array and valid check logic inside the VIP memory. In the VIP memory, every word has a 1-bit valid bit entry that is dynamically updated. The valid bit array is initialized when a processor resets or at every end of shared data transactions. By the write access from the producer PE, the valid bit of the corresponding address is set to HIGH. When an empty memory address with a LOW valid bit is accessed by the consumer PEs, the valid bit check logic asserts an INVALID signal to prevent reading false data. Figure 2.14 illustrates the overall procedure of the proposed memory transaction control. We assume again that PE 1 is the producer PE, and PEs 3 and 4 are consumer PEs. In the example data transaction, PE 3 reads the shared data at address 0×0 and PE 4 reads the shared data at address 0×8, whereas PE 1 writes the valid data only at address 0×0 of the VIP memory (Fig. 2.14(a)). Because the valid bit array has a HIGH bit for the address 0×0 only, the NIM of PE 4 obtains an INVALID packet instead of normal packets with valid data (Fig. 2.14(b)). Then, the NIM of PE 4 periodically retires reading valid data at address 0×8 until PE 1 also writes valid data at address 0×8 (Fig. 2.14(c)). Meanwhile, the operation of PE 4 is in a hold state. After reading the valid shared data from the VIP memory, the operation of the PE continues (Fig. 2.14(d)).

The advantages of the memory transaction control are reduced NoC traffic and PE activity, which contribute to a low-power operation. For consumer PE polls on the valid shared data, receiving INVALID notification rather than barrier value reduces the number of flits traversed through the NoC because the INVLAID notification does not have address/data fields. In addition, no polling loops are required for waiting valid data because the memory-centric NoC automatically blocks the

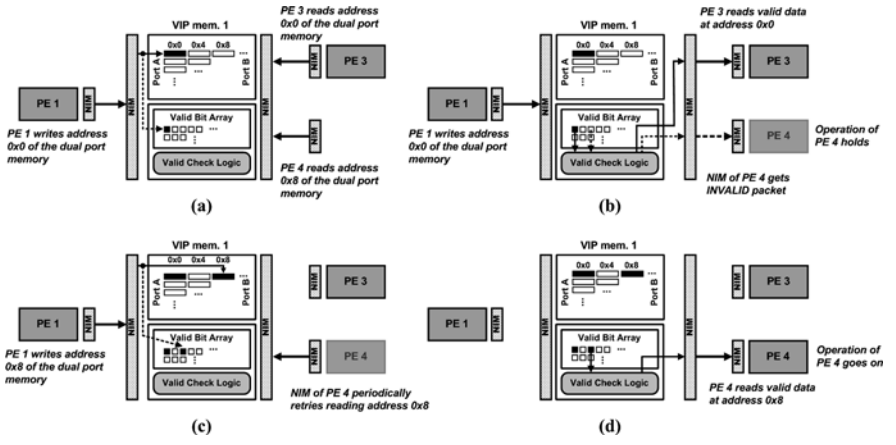


Fig. 2.14 Memory transaction control of the memory-centric NoC

access to the unwritten data. This results in reduced processor activity which is helpful for low-power consumption.

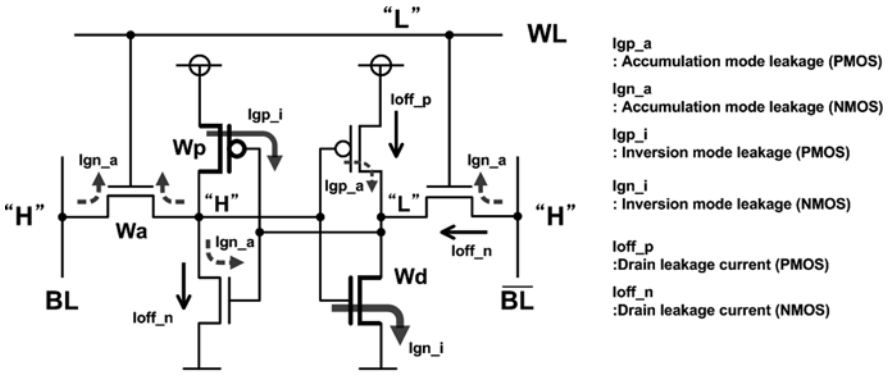
2.4 Low-Power Embedded Memory Design

At the start of this chapter, the concept of memory hierarchy was introduced first to draw a comprehensive map of memories in the computer architecture. After that, we discussed memory implementation techniques for low-power consumption regarding memory access patterns. Then, we discussed about the way of architecting the memory hierarchy considering the data flow of target applications for low-power consumption. As a wrap-up of this chapter, other low-power techniques applicable for memory design independent of data access pattern or data flow are introduced in this section. By using such techniques with application-specific optimizations, further reduction in power consumption can be achieved.

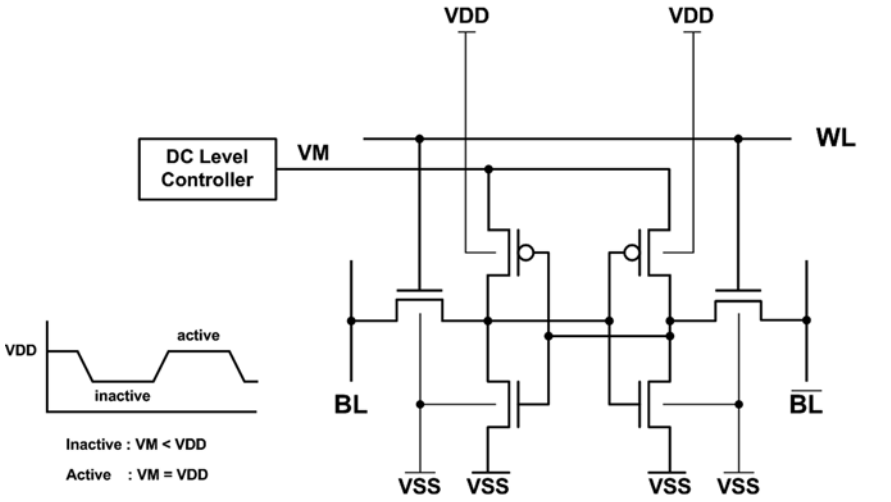
2.4.1 General Low-Power Techniques

For high-performance processors, providing data to be processed without bottleneck is as important as performing computation in high speed to achieve maximum performance. For that reason, there have been a number of researches for memory performance improvement and/or memory power reduction.

The common low-power techniques applicable to both DRAMs and SRAMs are summarized in [24]. This chapter reviews previously published low-power techniques such as reducing charge capacitance, operating voltage, and dc current, which focused on reducing power consumed by active memory operations. As the process technology has scaled down, however, static power consumption is becoming more and more important because the power dissipated due to leakage current of the on-chip memory starts to dominate the total power consumption in sub-micron process technology era. Even worse, the ITRS road map predicted that on-chip memory will occupy about 90% of chip area in 2013 [25] and this implies that the power issues in on-chip memories need be resolved. As a result, a number of low-power techniques for reducing leakage current in the memory cell have been proposed in recent decade. Koji Nii et al. suggested using lower NMOS gate voltage to reduce gate leakage current and peripheral circuits [26]. Based on the measured result that the largest portion of gate leakage current results from the turned on NMOS in the 6-transistor SRAM cell as shown in Fig. 2.15(a), controlling cell supply voltage is proposed. By lowering the supply voltage of the SRAM cells when the SRAM is in idle state, gate leakage current can be reduced without sacrificing the memory operation speed and this scheme is shown in Fig. 2.15(b). On the other hand, Rabiul Islam et al. proposed back-bias scheme to reduce sub-threshold leakage current of the SRAM cells [27]. This back-bias scheme is also applied when the SRAM is in idle state, and back-bias voltage is removed in normal operation.



(a) Leakage model in six-transistor SRAM cell



(b) Gate leakage current suppression in a cell

Fig. 2.15 Gate leakage model and suppression scheme [26]

More recent researches attempted to reduce leakage current more aggressively. Segmented virtual ground (SVGND) architecture was proposed to improve both static and dynamic power consumptions [28]. The SVGND architecture is shown in Fig. 2.16. The bit line of the SRAM is divided into M+1 segments, where each segment consists of a number of SRAM cells sharing the same segment virtual ground (SVG) and each SVG is switched between the real column virtual ground (CVG) and V_L voltage according to the corresponding segment select signals. In the SVGND architecture, only about 1/3–2/3 of power supply voltage is adaptively provided to the SRAM cells through the V_H and V_L signals instead of power and

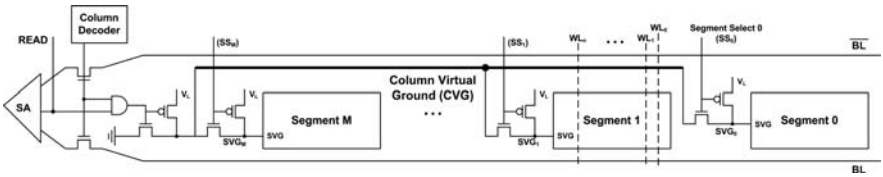


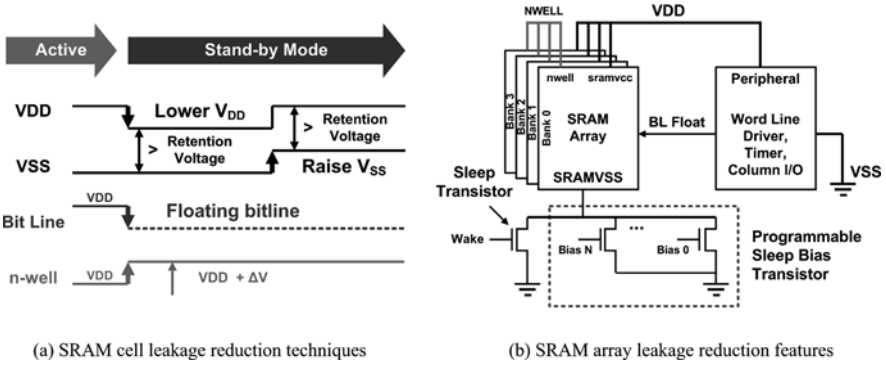
Fig. 2.16 Concept of SVGND SRAM [28]

ground signals, respectively. In this scheme the V_H is fixed and adjusted around two thirds of the supply voltage and the V_L is controlled between about one third of the supply voltage and the ground. At first, static power reduction is clearly visible. By reducing voltage across the SRAM cells, both gate and sub-threshold leakage currents can be kept in very low level. In addition, maintaining the source voltage of the NMOS (V_L) higher than its body bias voltage (Vss) has the effect of reverse biasing, and this results in further reduction of sub-threshold leakage current. The dynamic power consumption of the SRAM is also reduced by lower voltage across the SRAM cells. In the case of write operation, cross-coupled inverter chain in the SRAM cell can be driven to the desired value more easily. Compared to the SRAM cells with full supply voltages, the driving forces of SVGND SRAM cells have lower strength. When the read operation occurs, SVG line of each segment is pulled down to ground to facilitate sense amplifier operation. The power reduction in the read operation comes from selective discharge of SVG node, which prevents unnecessary discharge of internal capacitances of the neighboring cells in the same row.

As the process scales down to deep sub-micron, a more powerful leakage current reduction scheme is required. In the SRAM implementation using 65 nm process technology, Yih Wang et al. suggested using a series of leakage reduction techniques at the same time [29]. In addition to scaling of retention voltage in the SRAM cells, bit-line floating and PMOS back-gate biasing are also adopted. Lowering the retention voltage across the SRAM cell is the base for reducing gate and junction leakage current. However, there still remains junction leakage current from the bit line pre-charged to Vdd voltage which is higher than SRAM cell supply voltage. The bit-line floating scheme is applied to reduce the junction current through the gate NMOS. Finally, the PMOS back-gate biasing scheme suppresses leakage current through PMOS transistors in the SRAM cell which results from the lowered PMOS gate voltage due to retention voltage lowering. Figure 2.17 shows the concepts of leakage reduction schemes and their application to the SRAM architectures.

2.4.2 Embedded DRAM Design in RAMP-IV

Embedded memory design has advantages to system implementation. One of the biggest benefits is energy reduction in the memory interface and ease of memory utilization such as bus width. General system-on-board designs use off-the-shelf

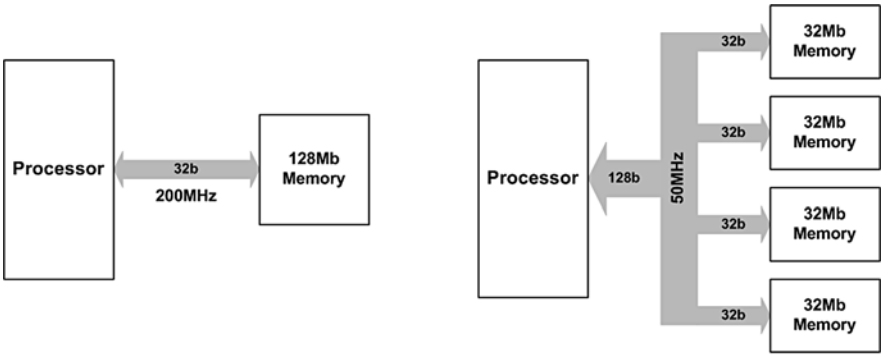


(a) SRAM cell leakage reduction techniques (b) SRAM array leakage reduction features

Fig. 2.17 SRAM cell leakage reduction techniques and their application to the SRAM [29]

memory devices and they have narrow bus width like 16 bit or 32 bit. For large data bandwidth, the system needs to increase the clock frequency or the bus width by using many memory devices in parallel. Figure 2.18 shows examples to realize 6.4 Gbps bandwidth. The first option increases the power consumption and the latter option occupies large system footprint. And both of them consume large power in pad drivers between the off chip memory and the processor. Embedded memory design, on the contrary, is free from the number of the bus width because interconnection between the memory and the processor inside the die occupies a little area. And the interconnection inside the die does not need large buffers like pad drivers. Another benefit is that the embedded memory does not need to follow conventional memory interface which is standard but somewhat redundant. Details will be shown through the 3D graphics rendering engine design with embedded DRAM memory.

Figure 2.19 shows the architecture of 3D graphics rendering processor [11]. Totally 29 Mb DRAMs are split into three memory modules: frame buffer, depth



(a) High Frequency Approach (b) Parallel Memory Approach

Fig. 2.18 Large bandwidth approaches for system-on-board design

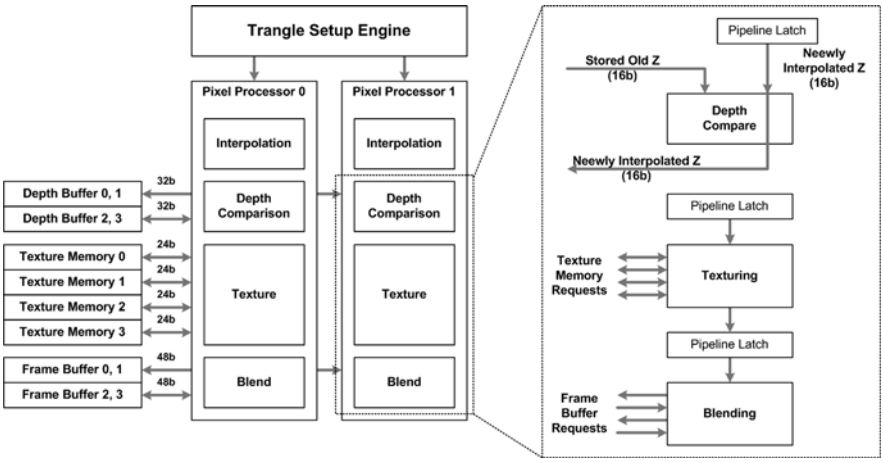


Fig. 2.19 Three-dimensional graphics rendering processor with DRAM-based EML approach

buffer, and texture memory. And each memory is physically divided into four memory modules so that totally 12 memory modules are used. In the pixel processors, scene or pixel is compared with the previous one. After that it is textured and blended in the pipeline stages. Each stage needs its own memory access to complete the rendering operations. And the memory access patterns are each different. For example, depth comparison and blending needs read–modify–write (RMW) operation while texturing needs just read operation. If the system is implemented by off-chip memories, the rendering processor needs 256 data pins and additional control signal pins, which cause increase in both package size and power consumption due to the pad driving. The processor shown in this example integrates the memories on a single chip and eliminates more than 256 pads for memory access.

For operation-optimized memory control, depth buffer and frame buffer are designed to support the single-cycle RMW operation with separate read and write buses, whereas the texture memory uses shared read/write bus. In order to provide the operation-optimized memory control for depth comparison and blending, the frame buffer and depth buffer support a single-cycle read–modify–write data transaction using separate read and write buses. It drastically simplifies the memory interface of the rendering engine and the pipeline, because the data required to process a pixel are read from the frame and depth buffers, calculated in the pixel processor, and written back to the buffers within a single clock period without any latency. Therefore, caching and pre-fetching, which may cause power and area overhead, are not necessary in the RMW-supporting architecture. The timing diagram of the RMW operation in the frame buffer is depicted in Fig. 2.20. To realize low-power RMW operation, command sequence is designed to use "PCG-ATV-READ-HOLD-WRITE" instead of "ATV-READ-HOLD-WRITE-PCG." In this case, the PCG sequence could be skipped in consecutive RMW operations. The write-mask signal, which is generated by the pixel processor, decides the activation of the write

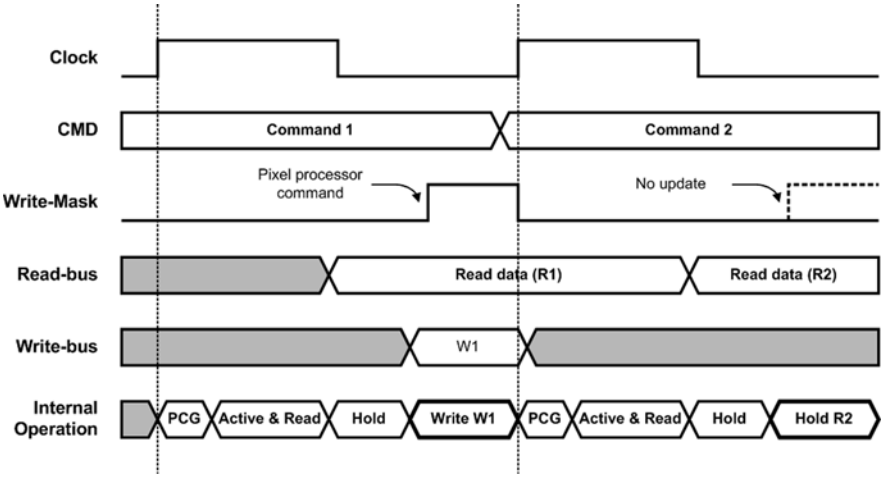


Fig. 2.20 Frame buffer access with read–modify–write scheme

operation. With single-cycle RMW operation, the processor does not need to use over-clocked frequency, and memory control is simple.

2.4.3 Combination of Processing Units and Memory – Visual Image Processing Memory

The other memory design technique for low-power consumption is to embed the processing units inside the memory. According to the target application domains, this technique is not always applicable in general. In the case of application-specific processor, however, the memory hierarchy is tailored for the target application and various types of memories are integrated together. Among them, some application-specific memories may incorporate the processing ability to improve overall performance of the processor. In case, large amount of data are loaded to a processor core from the memory and fixed operations are repeatedly performed on the loaded data, integrating the processing unit inside the memory is advantageous for removing the overhead of loading data into the processor core. A good implementation example of a memory with processing capability is a visual image processing (VIP) memory of KAIST [18, 30]. The VIP memory is briefly mentioned in Section 2.3 when describing the memory-centric NoC. Its function is to read out the address of local maximum pixel inside the 3×3 window in response to center pixel address of the $3\times$window.

The VIP memory has two behavioral modes: *normal* and *local-maximum* modes. In *normal* mode, VIP memory operates as a synchronous dual-port SRAM. It receives two addresses and control signals from the two ports and reads or writes two 32-bit data independently. While in *local-maximum* mode, the VIP memory finds

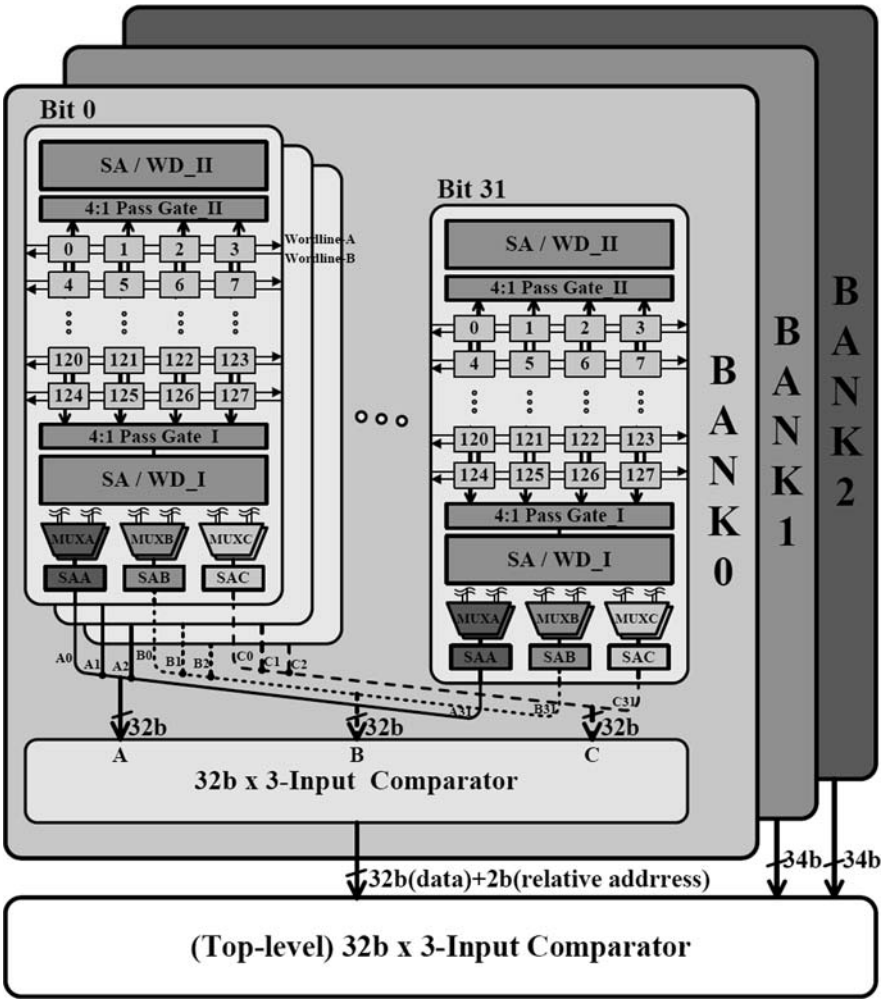


Fig. 2.21 Overall architecture of the VIP memory

the address of the local maximum out of the 3×data window when it receives the address of the center location of the window. Figure 2.21 shows the overall architecture of the VIP memory. It has a 1.5 KB capacity and consists of three banks. Each bank is composed of 32 rows and 4 columns and operates in a word unit. Each bit of four columns shares the same memory peripherals such as write driver and sense amplifier and the logic circuits for local maximum location search (LMLS). The LMLS logic is composed of multiplexers and tiny sense amplifiers. And a 3-input comparator for 32-bit number is embedded within the memory arrays.

Before the LMLS inside a 3×3 window, the pixel data of the image space should have been properly mapped into the VIP memory. First, the rows of the visual image

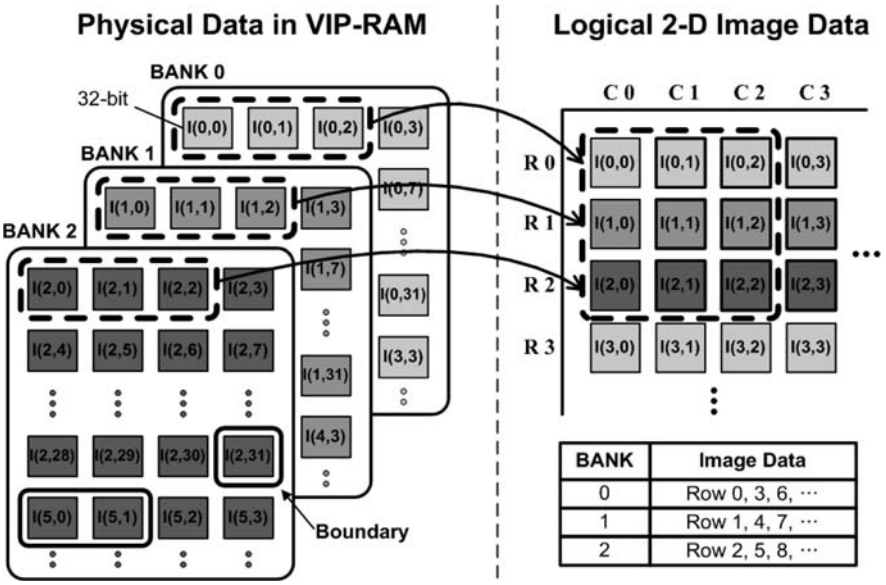


Fig. 2.22 Data arrangement in the VIP memory

data are interleaved into different banks according to the modulo-3 operation on the row number as shown in Fig. 2.22. Then, the three 32-bit data from the three banks form the 3×3 window. In the VIP memory with properly mapped data, LMLS operation is processed in three steps. First, two successive rows are activated and three corresponding data are chosen by multiplexers. Second, the 32-bit 3-input comparators in three banks deduce the three intermediate maximum values among the respective three numbers of the respective bank. Finally, the top level 32-bit 3-input comparator finds the final maximum value of the 3×3 window from three bank-level intermediate results and outputs the corresponding address.

The VIP memory is composed of dual-ported storage cell which has eight transistors, as shown in Fig. 2.23. A word line and a pair of bit lines are added to the conventional 6-transistor cell. Pull-down NMOS transistors are larger than other minimum-sized transistors for stability in data retention. A single cell layout occupies 2.92 μm × 5.00 μm in a 0.18-μm process. Bitwise competition logic (BCL) is devised to implement a fast, low-power, and area-efficient 32-bit 3-input comparator. It just locates the first "1" from the MSB to the LSB of two 32-bit numbers and decides the larger number between the two 32-bit binary numbers without complex logics. The BCL enables the 32-bit 3-input comparator to be compactly embedded in the memory bank. Figure 2.24 describes its circuit diagram and operation. Before input to BCL comparator, each bit of two numbers are pre-encoded from A[i] and B[i] into (A[i] · ∼B[i]) and (∼A[i] · B[i]), respectively. Pre-encoding prevents the occurrence of logic failures in the BCL when both inputs have 1 at the same bit position. In the BCL, A line and B line are pre-charged to VDD initially. Then,

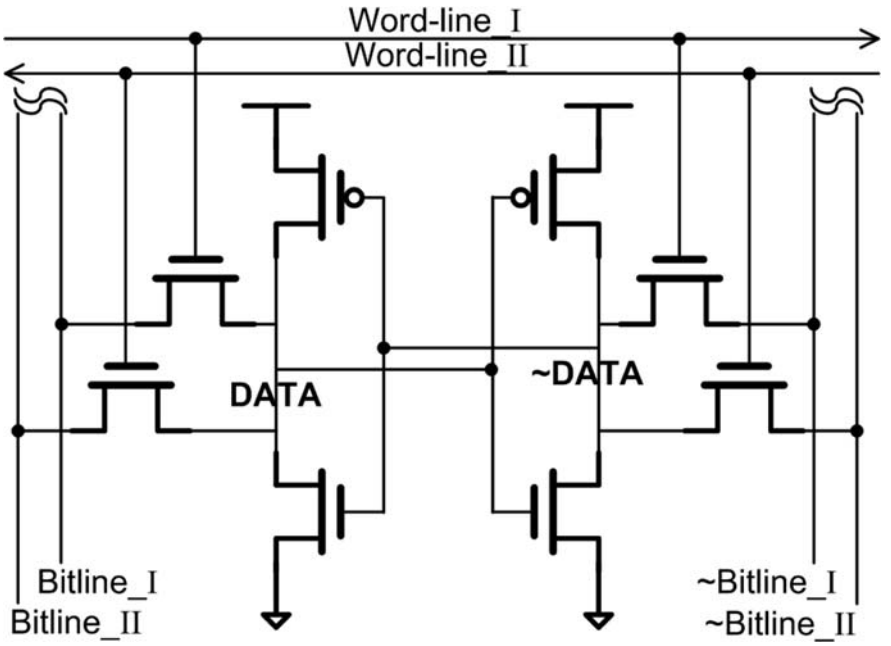
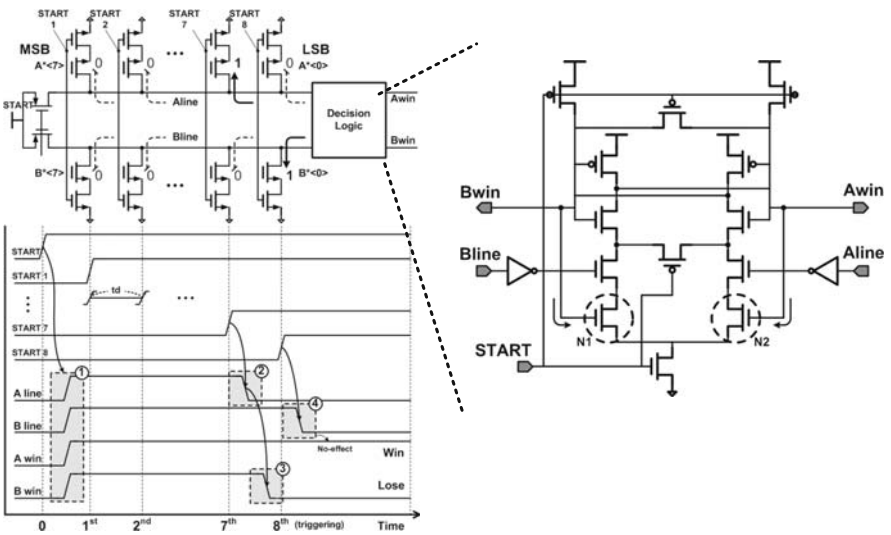


Fig. 2.23 A dual-ported memory cell of the VIP memory



(a) Bitwise Competition Logic Operation (b) Decision Logic of the BCL

Fig. 2.24 Bitwise competition logic of the VIP memory

START signals are activated to trigger each bit of pre-encoded signals sequentially from MSB to LSB. If any triggered signal is 1, the path from the corresponding line to GND is opened and its voltage goes down immediately. Then, decision logic, at the right end of the lines, detects the line that first goes down and keeps the result until the bit comparisons end. As shown in Fig. 2.24, the circuit of decision logic is the same as the sense amplifier except transistors N1 and N2. N1 and N2 receive the feedback signals and disable input of the small number to preserve only the first decision or the large number. For example, the timing diagram of Fig. 2.24 illustrates its operation in case that the two pre-encoded inputs A∗ and B∗ are 00000010 and 00000001. The gray boxes represent the transitions of a few important events in BCL operation. At box (1), all lines are pre-charged when the START signal is low. Triggering starts but both lines stay in VDD by the seventh start signal. At the seventh triggering of box (2), A line is dropped to GND because the seventh bit of A is 1. The drop of A line forces the decision logic to turn down *Bwin* signal of box (3). Finally, *Awin* and *Bwin*, which represent the comparison results, are kept until the end of the cycle irrespective of B line voltage as shown in box (4). The 32-bit data comparator is composed of four parallel 8-bit BCLs. The 32-bit comparison results can be obtained from the four results of the four parallel BCLs by setting higher priority to the result of the MSB part BCL. As a result, the 32-bit 2-input comparator with BCL uses only 482 transistors, which are 38% less than the transistor count of the comparator reported in [31]. The 32-bit 3-input comparator is designed using three of 32-bit 2-input BCL comparators. Its worst case delay is 1.4 ns, which is sufficiently small considering the 5-ns timing budget of VIPRAM when operating at 200 MHz.

References

1. Lu Peng, et al., "Memory Performance and Scalability of Intel's and AMD's Dual-Core Processors: A Case Study," IEEE International Performance, Computing, and Communication Conference, pp.55–64, April, 2007.
2. Dac C. Pham, et al., "Overview of the Architecture, Circuit Design, and Physical Implementation of a First-Generation Cell Processor," IEEE Journal of Solid-State Circuits, Vol. 41, Issue 1, pp.179–196, Jan. 2006
3. Marc Tremblay and Shailender Chaudhry, "A Third-Generation 65 nm 16-Core 32-Thread Plus 32-Scout-Thread CMT SPARC Processor," Technical Digest of IEEE International Solid State Circuits Conference, pp.82–83, February, 2008
4. Seung-Jun Bae, et al., "A 60 nm 6 Gb/s/pin GDDR5 Graphics DRAM with Multifaceted Clocking and ISI/SSN-Reduced Techniques," Technical Digest of IEEE International Solid State Circuits Conference, pp.82–83, February, 2008
5. Kyungwoo Nam, et al., "A 512 Mb 2-Channel Mobile DRAM (oneDRAM^{TM}) with Shared Memory Array," IEEE Asian Solid-State Circuits Conference, pp.204–207, November, 2007.
6. Samsung High Speed SRAM product page, http://www.samsung.com/global/business/semiconductor/products/sram/Products_HighSpeedSRAM.html, 2008
7. International Technology Roadmap for Semiconductors, Interconnect, 2003 Edition, Semiconductors Industry Assoc. and SEMATECH.
8. John L. Hennessy and David A. Patterson, "Computer Architecture – A Quantitative Approach," third edition, pp.390–392, San Francisco, USA, Morgan Kaufmann Publishers, 2003.

9. Satoru Tanoi, et al., “A 32 Bank 256 Mb DRAM with Cache and TAG,” Technical Digest of IEEE International Solid State Circuits Conference, Feb., 1994.
10. Barth J.E., Jr., et al., “A 500-MHz Multi-Banked Compliable DRAM Macro with Direct Write and Programmable Pipelining,” IEEE Journal of Solid-States Circuits, Vol. 40, Jan. 2005.
11. Ramchan Woo, et al., “A 210-mW Graphics LSI Implementing Full 3-D Pipeline with 264 Mtexels/s Texturing for Mobile Multimedia Applications,” IEEE Journal of Solid-States Circuits, Vol. 39, Feb., 2004
12. Marvell technology product page, http://www.marvell.com/products/cellular/applications.jsp, 2008.
13. Marvell technology, “PXA300 Product Brief,” http://www.marvell.com/files//products/cellular/ application/PXA300_PB_R4.pdf, 2008
14. Brucek Khailany, et al., “IMAGINE: Media Processing with Streams,” IEEE Micro, Volume 21, Issue 2, pp. 35–46, Mar.-Apr., 2001.
15. William J. Dally, et al., “Stream Processors: Programmability with Efficiency,” ACM Queue, Vol. 2, Issues 1, pp. 52–62, 2004
16. Donghyun Kim, et al., “Solutions for Real Chip Implementation Issues of NoC and Their Application to Memory-Centric NoC,” IEEE/ACM 1st International Symposium on Networks-on-Chip, pp. 30–39, May, 2007.
17. Donghyun Kim, et al., “An 81.6 GOPS Object Recognition Processor Based on NoC and Visual Image Processing Memory,” IEEE Custom Integrated Circuits Conference, pp. 443–446, Sept. 2007.
18. David G. Lowe, “Distinctive Image Features from Scale-Invariant Key points,” ACM Intl. Journal of Computer Vision, Vol. 60, Issue 2, pp. 91–110, 2004.
19. Sunghwan Ahn, et al., “Data Association Using Visual Object Recognition for EKF-SLAM in Home Environment,” Proceedings of IEEE Intl. Conf. on Intelligent Robots and Systems, pp. 2760–2765, 2006.
20. Patric Jensfelt, et al., “Augmenting SLAM with Object Detection in a Service Robot Framework,” IEEE Intl. Symposium on Robot and Human Interactive Communication, pp. 741–746, 2006.
21. Bertolli F., Jensfelt P., Christensen H.I., ”SLAM using Visual Scan-Matching with Distinguishable 3D Points,” IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4042–4047, Oct., 2006.
22. Zhang Nan, Li Maohai, Hong Bingrong, ”Active Mobile Robot Simultaneous Localization and Mapping,” IEEE International Conference on Robotics and Biomimetics, pp. 1671–1681, Dec., 2006.
23. Kangmin Lee, Se-Joong Lee and Hoi-Jun Yoo, “Low-power network-on-chip for high-performance SoC design,” IEEE Transactions on Very Large Scale Integration Systems, Vol. 14, Issue 2, pp. 148–160, Feb., 2006.
24. Kiyoo Itoh, Katsuro Sasaki, and Yoshinobu Nakagome, “Trends in Low-Power RAM Circuit Technologies,” IEEE Digest of Technical Papers of Symposium on Low Power Electronics, pp. 84–87, Oct., 1994.
25. International Technology Roadmap for Semiconductors, 2001 Update, Semiconductors Industry Assoc. and SEMATECH.
26. Koji Nii, et al., “A 90-nm Low-Power 32-kB Embedded SRAM with Gate Leakage Suppression Circuit for Mobile applications,” IEEE Journal of Solid-State Circuits, Vol. 39, Issue 4, pp. 684–693, Apr., 2004.
27. Rabiul Islam, Adam Brand, and Dave Lippincott, “Low Power SRAM Techniques for Handheld Products,” IEEE Proceedings of the 2005 International Symposium on Low Power Electronics and Design (ISLPED), pp. 198–202, Aug., 2005.
28. Mohammad Sharifkhani, and Majog Sachdev, “Segmented Virtual Ground Architecture for Low-Power Embedded SRAM,” IEEE Transactions on Very Large Scale Integration Systems (TVLSI), pp. 196–205, Feb., 2007.

29. Yih Wang, et al., “A 1.1 GHz 12 uA/Mb-Leakage SRAM Design in 65 nm Ultra-Low-Power CMOS Technology with Integrated Leakage Reduction for Mobile Applications,” IEEE Journal of Solid-State Circuits, Vol. 43, Issue 1, pp. 172–179, Jan., 2008.
30. Joo-Young Kim, et al., “Visual Image Processing RAM for Fast 2-D Data Location Search,” IEEE European Solid State Circuits Conference, pp. 324–327, Sept., 2007.
31. Shun-Wen Cheng, “A High-Speed Magnitude Comparator with Small Transistor Count,” IEEE Proceedings of International Conference on Electronics, Circuits and Systems, Vol. 3, pp. 1168–1171, Dec., 2003.